

```

// pushcons.C
// Copyright (c) 2003. Borland Software Corporation, All Rights Reserved.
void
VISGIOPProxyTypedPushConsumer::invoke(
    CORBA::ServerRequest_ptr req)
{
    // a remote pushing
    EventWrapper* wrap = _manager->create(req->strm(), req->op_name());
    push(wrap);
    EventWrapper::_release(wrap);
    return ;
}

// pushsup.C
// Copyright (c) 2003. Borland Software Corporation, All Rights Reserved.
void
VISGIOPProxyPushSupplier::_push_to_target(
    EventWrapper* wrap,
    CORBA::UShort lowdeg)
{
    VISGlobalTable* tls = VISGlobalTable::instance();
    tls->byte_order = wrap->_payload_byte_order;
    tls->payload_curoff = wrap->_payload_curoff;
    CORBA::MarshalOutBuffer_var obuf;
    obuf = _consumer->_vis_request(wrap->get_operation(), 0x01);
    obuf->put(wrap->_payload_ptr, wrap->_payload_len);
    CORBA::MarshallInBuffer_var ibuf;
    ibuf = _consumer->_invoke(obuf);
    tls->payload_curoff = 0UL;
}

// vgconnect.C.txt
// Copyright (c) 2003. Borland Software Corporation, All Rights Reserved.
//
// Code from vgconnect.C. Scenario A.
//
GIOP::Version& VISGIOPProtocolConnector::get_giop_version()
{
    if(VISGlobalTable::is_not_visinotify()) {
        assert(_profile);
        return _profile->version();
    }
    // For scenario A:
    // Cases we need to use lower giop version
    // 1. The consumer is in low version.
    // 2. The received payload isn't aligned on 8.
    //
    GIOP::Version& version = _profile->version();
    if( version.minor < 0x02 ) {
        // case 1.
        return version;
    }
}

```

```

VISGlobalTable* tIs = VISGlobalTable::instance();
if( tIs->payload_curoff%8 ) {
    // case 2.
static GIOP::Version v1_0 = { 0x01, 0x00 };
return v1_0;
}
// either normal request (i.e. payload_curoff == 0) or
// it already aligned on 8.
return version;
}
// vgmsg.C.txt
// Copyright (c) 2003. Borland Software Corporation, All Rights Reserved.
//
// Code from vgmsg.C. Scenario A.
//
void VISGIOPRequest::_marshal_out_1_x()
{
    if(!VISGlobalTable::use_local_byte_order()) {
        VISGlobalTable* tIs = VISGlobalTable::instance();
        this->byte_order(tIs->byte_order);
    }
    VISGIOPOutputBuffer& obuf = _output;
    // marshal the header
    obuf << _service_context;
    obuf << _request_id;
    obuf << _response_expected;
    obuf << _object_key;
    obuf << _operation;
    // write out principal,
    if(VISGlobalTable::is_not_visinotify()) {
        // normally, always zero length principal
        obuf << (CORBA::ULong)0UL;
    }
    else {
        // For visinotify -Ke
        VISGlobalTable* tIs = VISGlobalTable::instance();
        if( tIs->payload_curoff == 0UL ) {
            // normal request -- still zero length principal
            obuf << (CORBA::ULong)0UL;
        }
        else {
            // visinotify channel pushing request, use principal to adjust
            // alignment. Scenario A.
            obuf.align(4); // align for principal count.
            CORBA::Long delta =
                (tIs->payload_curoff)%8UL - (obuf.curoff() + 4UL)%8UL;
            delta = (8L+delta)%8L;
            obuf << (CORBA::ULong)delta;
            if( delta ) {
                static char princ_data[7] =

```

```

    {0x00,0x00,0x00,0x00,0x00,0x00,0x00};
    obuf.put(princ_data, delta);
}
}
}
// Isilva - set dataOffset
obuf.dataOffset(obuf.curoff());
}
// VISGIOPReply.txt
// Copyright (c) 2003. Borland Software Corporation, All Rights Reserved.
VISGIOPReply::VISGIOPReply(CORBA::UShort version,
                           CORBA::ULong request_id,
                           const IOP::ServiceContextList& service_context,
                           ProtocolEngine::ReplyStatus reply_status)
: VISGIOPFragmentable(version, GIOP::Reply),
  VISGIOPMessage(version, GIOP::Reply, MSG_FRAGMENTABLE),
  _reply_status(reply_status)
{
    _service_context = service_context;
    _request_id = request_id;
    CORBA::ULong payload_curoff = 0UL;
    if(!VISGlobalTable::use_local_byte_order()) {
        VISGlobalTable* tls = VISGlobalTable::instance();
        this->byte_order(tls->byte_order);
        payload_curoff = tls->payload_curoff;
    }
    assert(_output);
    VISGIOPOutputBuffer& obuf = _output;
    // write out the reply
    if (version == VISGIOPMessage::VERSION_1_0 ||
        version == VISGIOPMessage::VERSION_1_1) {
        // GIOP 1.0 or 1.1 write
        CORBA::ULong offset = obuf.curoff(); // mark the offset
        obuf << _service_context;
        if( payload_curoff != 0UL ) {
// for visinotify -Ke
obuf.align(4);
if(payload_curoff%8 != obuf.curoff()%8) {
//
// The payload could be from a 1.2 supplier or
// a 1.0/1.1 supplier. When it is from 1.0/1.1
// supplier, we requires its payload must be
// aligned on either 4 or 8. As long as application
// doesn't use GIOP 1.0/1.1 principal (i.e. principal
// is zero-length), this condition is always met.
//
CORBA::ULong len = _service_context.length();
_service_context.length(len + 1);
IOP::ServiceContext& sc = _service_context[len];
sc.context_id = 141000L; // dummy, todo: pick up a id

```

```

    // from our assigned range.
static CORBA::Octet ctx[4] = { 0x00, 0x00, 0x00, 0x00 };
sc.context_data.replace(4, 4, ctx);
obuf.seekpos(offset); // go back to the marked position
obuf << _service_context; // remarshal it to cause a
    // extra 12 bytes added into
    // the message to adjust the
    // payload start point.
// _service_context.length(len); // don't need to
    // recover the length
}
}
obuf << request_id;
obuf << reply_status;
} else {
    // GIOP 1.2
    obuf << request_id;
    obuf << reply_status;
    obuf << _service_context;
    // pad to next 8-byte boundary. See CORBA2.3 specification, section
    // 15.4.2.2
    obuf.do_data_alignment_padding(version);
}
// lsilva - set dataOffset
obuf.dataOffset(obuf.curoff());
}

```